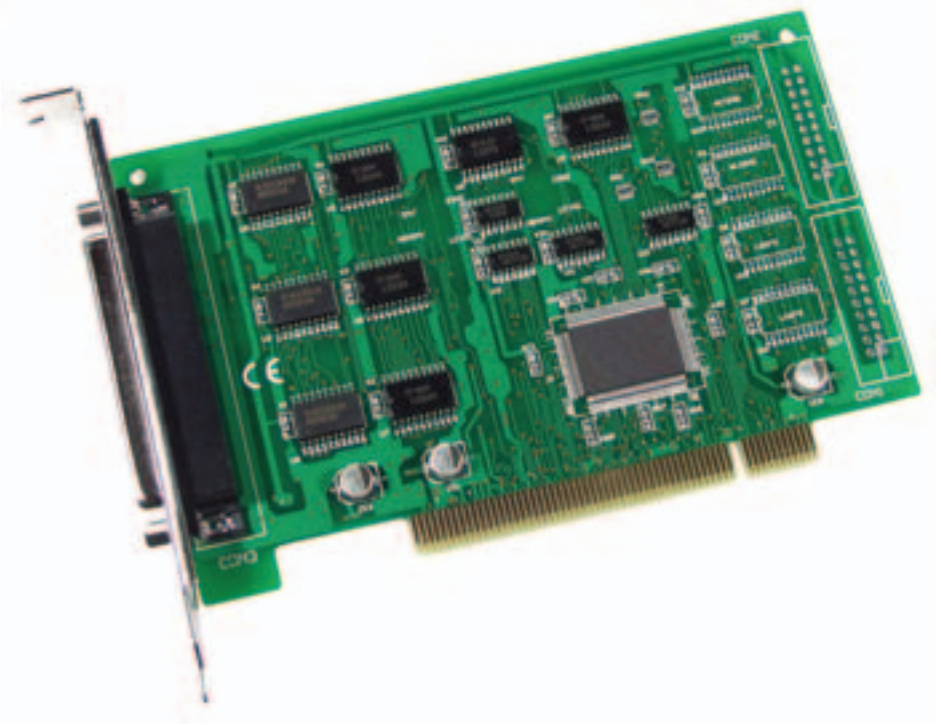
User's Guide



OME-PIO-D56/D24

PCI-Bus Digital I/O Board

Hardware Manual

omega.com®
OMEGA®

OMEGAnet® Online Service www.omega.com	Internet e-mail info@omega.com

Servicing North America:

USA:
ISO 9001 Certified
One Omega Drive, P.O. Box 4047
Stamford CT 06907-0047
TEL: (203) 359-1660 FAX: (203) 359-7700
e-mail: info@omega.com

Canada:
976 Bergar
Laval (Quebec) H7L 5A1, Canada
TEL: (514) 856-6928 FAX: (514) 856-6886
e-mail: info@omega.ca

For immediate technical or application assistance:

USA and Canada: Sales Service: 1-800-826-6342 / 1-800-TC-OMEGA®
Customer Service: 1-800-622-2378 / 1-800-622-BEST®
Engineering Service: 1-800-872-9436 / 1-800-USA-WHEN®
TELEX: 996404 EASYLINK: 62968934 CABLE: OMEGA

Mexico: En Español: (001) 203-359-7803 e-mail: espanol@omega.com
FAX: (001) 203-359-7807 info@omega.com.mx

Servicing Europe:

Benelux:
Postbus 8034, 1180 LA Amstelveen, The Netherlands
TEL: +31 (0)20 3472121 FAX: +31 (0)20 6434643
Toll Free in Benelux: 0800 0993344
e-mail: sales@omegaeng.nl

Czech Republic:
Frystatska 184, 733 01 Karviná, Czech Republic
TEL: +420 (0)59 6311899 FAX: +420 (0)59 6311114
Toll Free: 0800-1-66342 e-mail: info@omegashop.cz

France:
11, rue Jacques Cartier, 78280 Guyancourt, France
TEL: +33 (0)1 61 37 29 00 FAX: +33 (0)1 30 57 54 27
Toll Free in France: 0800 466 342
e-mail: sales@omega.fr

Germany/Austria:
Daimlerstrasse 26, D-75392 Deckenpfronn, Germany
TEL: +49 (0)7056 9398-0 FAX: +49 (0)7056 9398-29
Toll Free in Germany: 0800 639 7678
e-mail: info@omega.de

United Kingdom:
ISO 9002 Certified
One Omega Drive, River Bend Technology Centre
Northbank, Irlam, Manchester
M44 5BD United Kingdom
TEL: +44 (0)161 777 6611 FAX: +44 (0)161 777 6622
Toll Free in United Kingdom: 0800-488-488
e-mail: sales@omega.co.uk

It is the policy of OMEGA to comply with all worldwide safety and EMC/EMI regulations that apply. OMEGA is constantly pursuing certification of its products to the European New Approach Directives. OMEGA will add the CE mark to every appropriate device upon certification.

OME-PIO-D56/D24

User Manual

Table of Contents

1. Introduction

The OME-PIO-D56/OME-OME-PIO-D24 provides 56/24 TTL digital I/O lines. The OME-PIO-D56/OME-OME-PIO-D24 consists of one 24-bit bi-directional port, one 16-bit input port and one 16-bit output port (only for OME-PIO-D56). The 24-bit port supports three 8-bit groups PA, PB & PC. Each 8-bit group can be individually configured to function as either an input or an output. All groups using 24-bit bi-directional ports are configured as inputs upon power-up or reset.

Use the OME-DB-24PD to connect the input port for either isolation purposes, or to interface to the output port for relay control. The OME-PIO-D56/OME-PIO-D24 has one D-sub connector and two 20-pin flat-cable connectors (only for OME-PIO-D56). The flat cable can be connected to an OME-ADP-20/PCI adapter. The adapter can be fixed on the chassis. It can be installed in a 5V PCI bus and supports "Plug & Play".

1.1 Features

- PCI bus
- Up to 56/24(OME-PIO-D56/OME-PIO-D24) channels of digital I/O
- All I/O lines buffered on the board
- Eight-bit groups independently selectable for I/O on 24-bit port
- Input / Output programmable I/O ports under software control
- Double side SMD, short card.
- Connects directly to OME-DB-24PR, OME-DB-24PD, OME-DB-24RD, OME-DB-24PRD, OME-DB-16P8R, OME-DB-24POR, OME-DB-24SSR or OME-DB-24C
- 4 interrupt sources: PC0, PC1, PC2, PC3
- One DB37 connector, two 20-pin flat-cable connectors (only for OME-PIO-D56)
- High drive capability
- Automatically detected by Windows 95/98/2000/XP
- No base address or IRQ switches to set

1.2 Specifications

- **All inputs are TTL compatible**
  Logic high voltage : 2.4V (Min.)
  Logic low voltage : 0.8V (Max.)
- **All outputs are TTL compatible**
  OPTO-22 output (CON3)
  Sink current : 64mA (Max.)
  Source current : 32mA(Max.)
  16-channel output (CON1)
  Sink current : 8mA (Max.)
  Source current : 0.4mA(Max.)
- Environmental :
  Operating Temperature: 0°C to 60°C
  Storage Temperature: -20°C to 80°C
  Humidity: 0 to 90% non-condensing
- Dimensions: 143mm X 105mm
- Power Consumption: +5V @ 530mA/420mA(OME-PIO-D56/OME-PIO-D24)

1.3 Order Description

- OME-PIO-D56 : PCI bus 56-bit DI/O board
- OME-PIO-D24 : PCI bus 24-bit DI/O board

1.3.1 Options

- OME-DB-24PD : 24 channel isolated D/I board
- OME-DB-24RD : 24 channel relay board
- OME-DB-24PRD : 24 channel power relay board
- OME-DB-16P8R : 16 channel isolated D/I and 8 channels relay output board
- OME-DB-24POR : 24 channel Photo MOS output board
- OME-DB-24C : 24 channel open-collector output board
- OME-ADP-20/PCI : extender, 20-pin header to 20-pin header for PCI bus I/O boards

1.4 PCI Data Acquisition Family

We provide a family of PCI bus data acquisition cards. These cards can be divided into three groups as follows:

1. **OME-PCI-series: first generation, isolated or non-isolated cards**
   OME-PCI-1002/1202/1800/1802/1602: multi-function family, non-isolated
   OME-PCI-P16R16/P16C16/P16POR16/P8R8: D/I/O family, isolated
   OME-PCI-TMC12: timer/counter card, non-isolated
2. **OME-PIO-series: cost-effective generation, non-isolated cards**
   OME-PIO-823/821: multi-function family
   OME-PIO-D144/D96/D64/D56/D48/D24: D/I/O family
   OME-PIO-DA16/DA8/DA4: D/A family
3. **OME-PISO-series: cost-effective generation, isolated cards**
   OME-PISO-813: A/D card
   OME-PISO-P32C32/P64/C64: D/I/O family
   OME-PISO-P8R8/P8SSR8AC/P8SSR8DC: D/I/O family
   OME-PISO-730: D/I/O card
   OME-PISO-DA2: D/A card

1.5 Product Checklist

In addition to this manual, the package includes the following items:

- One OME-PIO-D56(or OME-PIO-D24) card
- One software floppy diskette or CD
- One release note

Please read the release note first. Important information that could be given in release note such as:

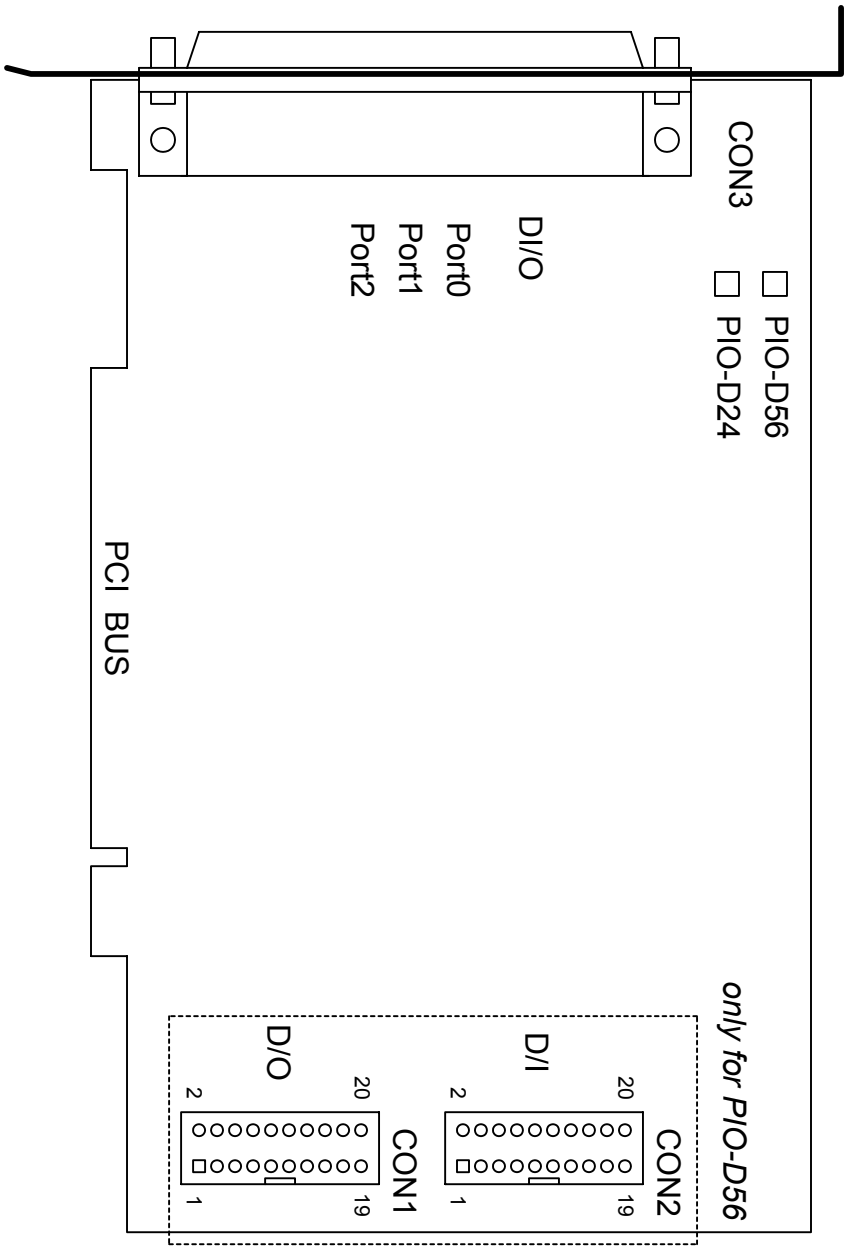
1. Where you can find the software driver & utility?
2. How to install software & utility?
3. The location of the diagnostic program?
4. FAQ

Attention!

If any one of these items is missing or damaged, please contact Omega Engineering immediately. Save the shipping materials and carton in case you want to ship or store the product in the future.

2. Hardware configuration

2.1 Board Layout



2.2 I/O Port Location

The OME-PIO-D56/OME-PIO-D24 consists of one 24-bit bi-directional port, one 16 bit input port and one 16 bit output port (only for OME-PIO-D56). The 24-bit port supports three 8-bit groups: PA, PB & PC. Each 8-bit group can be individually configured to function as either inputs or outputs. All groups using 24-bit bi-directional ports are configured as inputs upon power-up or reset. The I/O port locations are as follows:

Connector of OME-PIO-D56/D24	PA0 ~ PA7	PB0 ~ PB7	PC0 ~ PC7
CON3 (DI/O)	Port0	Port1	Port2

Connector of OME-PIO-D56	Description
CON1	D/O
CON2	D/I

Refer to Sec. 2.1 for board layout & I/O port location.

Note: PC0, PC1, PC2 and PC3 can be used as interrupt signal source. Refer to Sec. 2.4 for more information.

2.3 Enabling I/O Operation

2.3.1 DI/DO Port Architecture (CON3)

Upon power-up, all D/I/O port (CON3) operations are disabled. The RESET\ signal controls the enable/disable state of D/I/O port. Refer to Sec. 3.3.1 for more information about RESET\ signal. The power-up states are as follows:

- All D/I/O operations are disabled
- All three D/I/O ports are configured as D/I port
- All D/O latch register are undefined.(refer to Sec. 2.3.2)

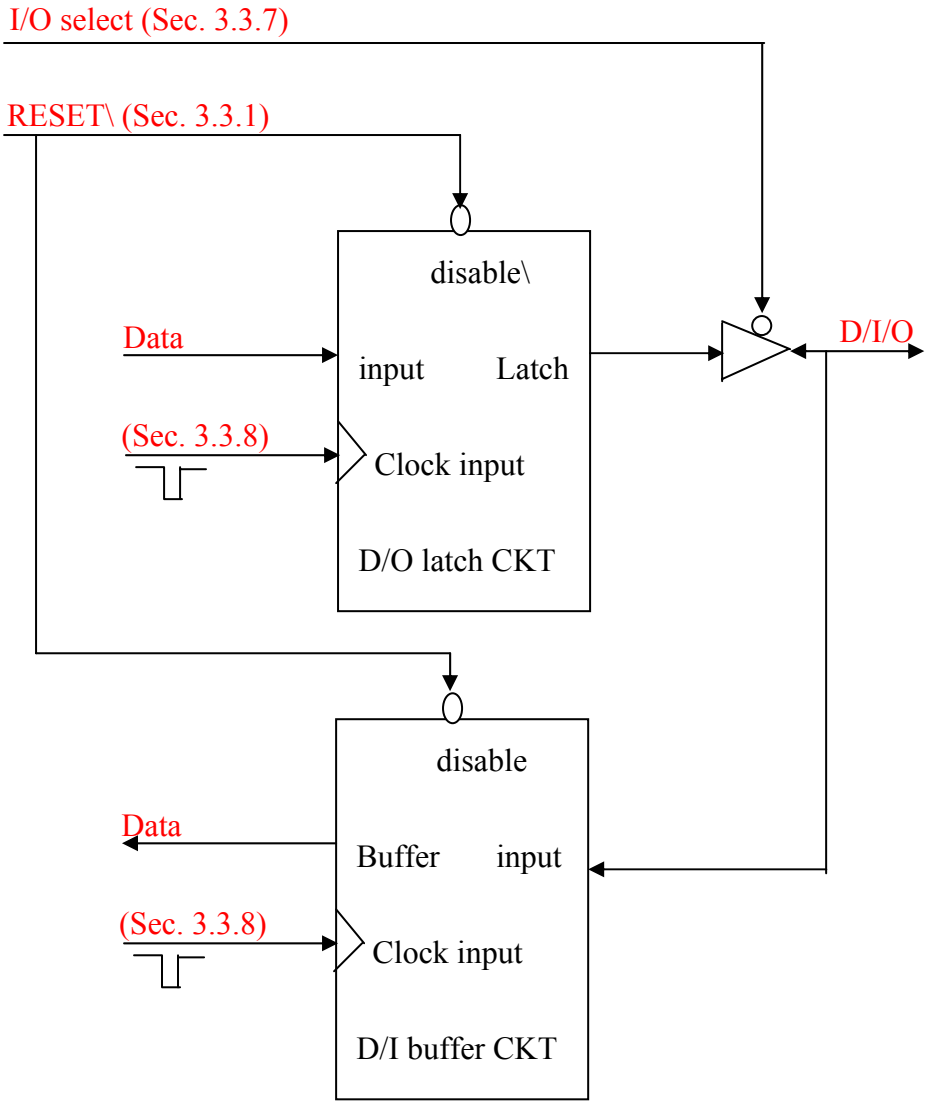
Initialization must be performed before using these D/I/Os. The recommended steps are as follows:

Step 1: Find address-mapping of OME-PIO/PISO cards (refer to Sec. 3.1)

Step 2: Enable all D/I/O operations (refer to Sec. 3.3.1)

Step 3: Configure the three ports (in CON3) to their expected D/I/O state & send the initial value to all D/O ports (refer to Sec. 3.3.8)

Refer to DEMO1.C for demo program.

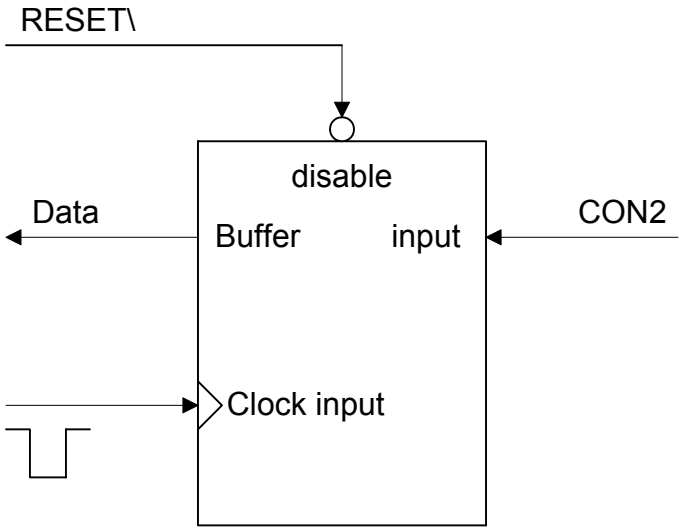


- When the RESET\ is in Low-state → all D/I/O operations are disabled
- When the RESET\ is in High-state → all D/I/O operation are enabled.
- If D/I/O is configured as D/I port → D/I= external input signal
- If D/I/O is configured as D/O port → D/I = read back of D/O
- If D/I/O is configured as D/I port → send to D/O will change the D/O latch register only. The D/I & external input signals will not change.

2.3.2 DI Port Architecture (CON2)

When the PC is powered up, all DI (CON2) port operations are disabled. The RESET\ signal controls the enable/disable signal for the DI port. Refer to Sec. 3.3.1 for more information about the RESET\ signal.

- The RESET\ is in Low-state → all DI operations are disabled
- The RESET\ is in High-state → all DI operations are enabled



D/I buffer CKT

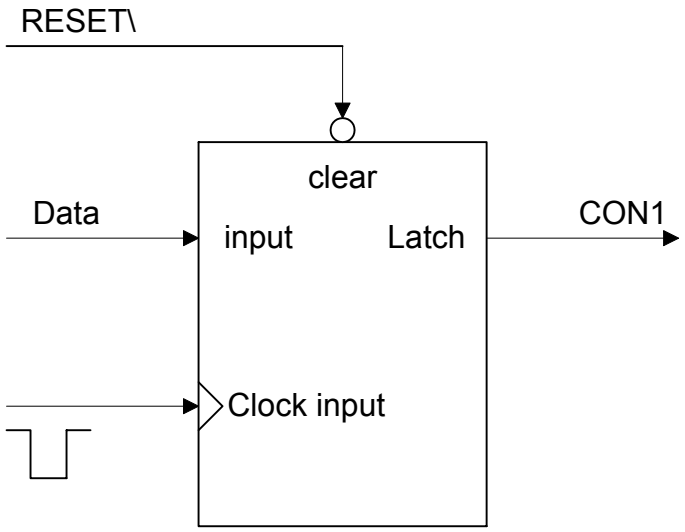
2.3.3 DO Port Architecture (CON1)

When the PC is powered up, all DO port (CON1) operations are disabled. The RESET\ signal controls the enable/disable signal for the DI port. Refer to Sec. 3.3.1 for more information about the RESET\ signal.

- The RESET\ is in Low-state → all DO operations are disabled
- The RESET\ is in High-state → all DO operations are enabled

The power-up states are as follows:

- All DO operations are disabled
- All output latches are cleared to Low-Level



D/O buffer CKT

2.4 Interrupt Operation

All PC0, PC1, PC2 and PC3 can be used as an interrupt signal sources. Refer to Sec. 2.1 for PC0/PC1/PC2/PC3 location. **The interrupt of OME-PIO-D56/OME-PIO-D24 is level-trigger & Active_High**. The interrupt signal can be programmed to **inverted or non-inverted** state. The programming procedure is given as follows:

1. Make sure **the initial level is High or Low**
2. If the initial state is High → select the **inverted** signal (Sec. 3.3.6)
3. If the initial state is Low → select the **non-inverted** signal (Sec. 3.3.6)
4. Enable the INT function (Sec. 3.3.4)
5. If the interrupt signal is active → program will transfer into the interrupt service routine → **if INT signal is High now → select the inverted input**
   → if INT signal is Low now → select the non-inverted input

Refer to DEMO3.C & DEMO4.C for single interrupt source. Refer to DEMO5.C for four interrupt sources.

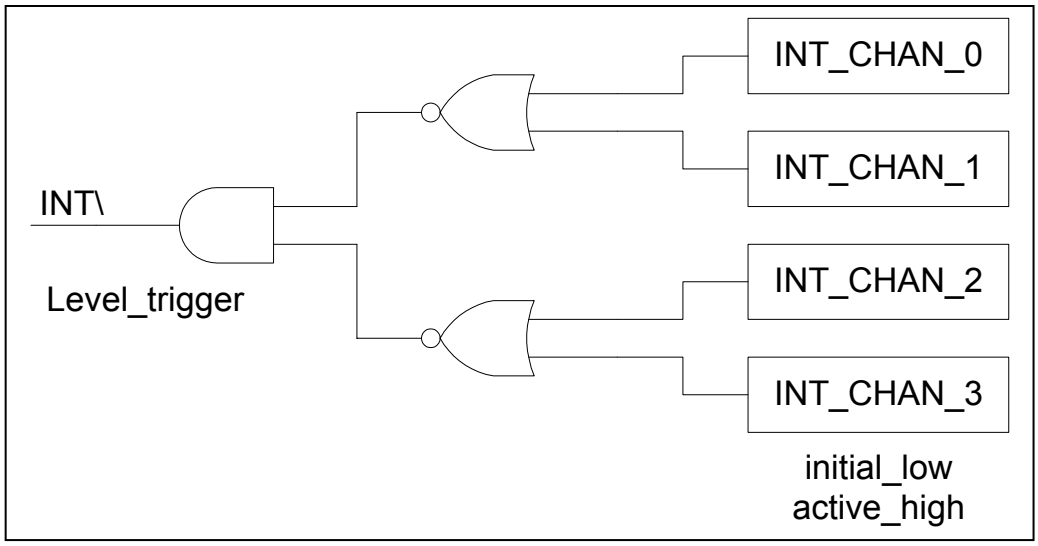
If only one interrupt signal source is used, the interrupt service routine does not have to identify the interrupt source. (Refer to DEMO3.C & DEMO4.C)

If there is more than one interrupt source, the interrupt service routine has to identify the active signals via the following steps: (Refer to DEMO5.C)

1. Reads the new status of the interrupt signal source
2. Compares the new status with the old status to identify the active signals
3. If PC0 is active, service PC0 & non-inverter/inverted the PC0 signal
4. If PC1 is active, service PC1 & non-inverted/inverted the PC1 signal
5. If PC2 is active, service PC2 & non-inverted/inverted the PC2 signal
6. If PC3 is active, service PC3 & non-inverted/inverted the PC3 signal
7. Saves the new status to old status

Note: If the interrupt signal is too short, the new status may be the same as old status. So the interrupt signal must be held active until the interrupt service routine is executed. This hold time is different for different operating systems. It can be as a short as micro-second or as a long as second. In general, 20ms is enough for most operating systems.

2.4.1 Interrupt Block Diagram of OME-PIO-D56/D24



The interrupt output signal of OME-PIO-D56/OME-PIO-D24, INT\ is Level_trigger & Active_Low. If the INT\ generates a low pulse, the OME-PIO-D56/OME-PIO-D24 will interrupt the PC only once. If the INT\ is fixed in low level, the OME-PIO-D56/OME-PIO-D24 will interrupt the PC continuously.
INT_CHAN_0/1/2/3 must be controlled in a pulse type signals. It must be fixed in low level state normally and generate a high pulse to interrupt the PC.

The priority of INT_CHAN_0/1/2/3 is the same. If all these four signals are active at the same time, then INT\ will be active only one time. So the interrupt service routine has to read the status of all interrupt channels for a multi-channel interrupt. Refer to Sec. 2.4 for mare information.

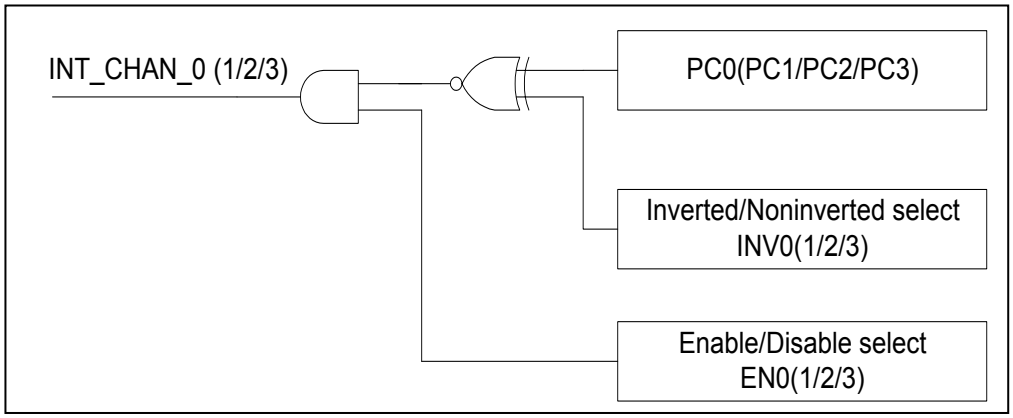
DEMO5.C → for multi-channel interrupt source

If only one interrupt source is used, the interrupt service routine does not have to read the status of interrupt source. The demo programs DEMO3.C and DEMO4.C are designed for single-channel interrupt demo, as follows:

DEMO3.C → for INT_CHAN_0 only (PC0 initial low)

DEMO4.C → for INT_CHAN_0 only (PC0 initial high)

2.4.2 INT_CHAN_0/1/2/3



The INT_CHAN_0 must normally be fixed in low level state and generate a high pulse to interrupt the PC.

The EN0 (EN1/EN2/EN3) can be used to enable/disable the INT_CHAN_0(1/2/3) as follows : (Refer to Sec. 3.3.4)

EN0 (1/2/3) = 0 → INT_CHAN_0(1/2/3) = disable

EN0 (1/2/3) = 1 → INT_CHAN_0(1/2/3) = enable

The INV0 can be used to invert/non-invert the PC0 (1/2/3) as follows: (Refer to Sec.3.3.6)

INV0 (1/2/3) = 0 → INT_CHAN_0(1/2/3) = inverted state of PC0 (1/2/3)

INV0 (1/2/3) = 1 → INT_CHAN_0(1/2/3) = non-inverted state of PC0 (1/2/3)

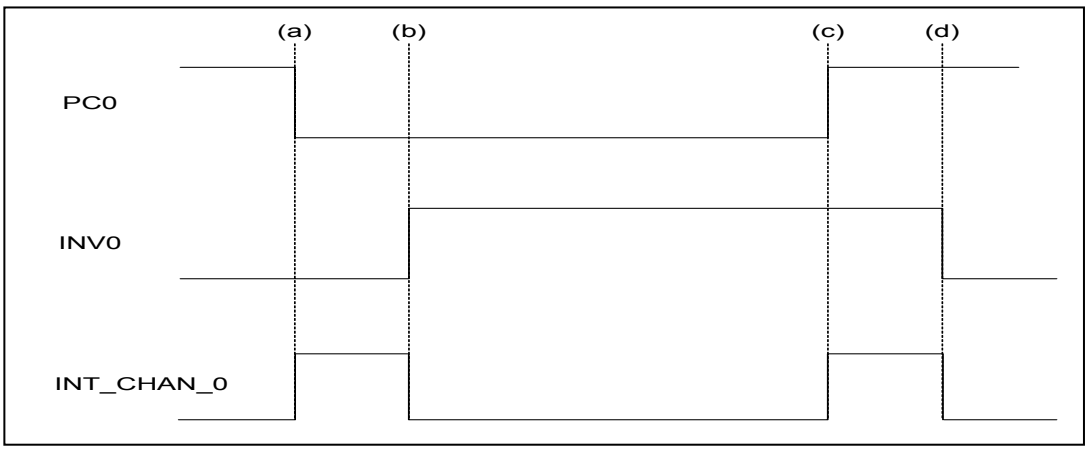
2.4.3 Initial_high, active_low Interrupt source

If the PC0 is a initial_high, active_low signal, the interrupt service routine should use INV0 to invert/non-invert the PC0 for high_pulse generation as follows: (Refer to DEMO4.C)

Initial setting:

```
now_int_state=1;            /* initial state for PC0       */
outportb(wBase+0x2a,0);  /* select the inverted PC0 */
```

```
void interrupt irq_service()
{
if (now_int_state==1)       /* now PC0 is changed to LOW          */(a)
   {                        /* --> INT_CHAN_0=!PC0=HIGH now       */
   COUNT_L++;               /* find a LOW_pulse (PC0)             */
   If((inport(wBase+7)&1)==0)/* the PC0 is still fixed in LOW     */
     {                      /* → need to generate a high_pulse   */
     outportb(wBase+0x2a,1);/* INV0 select the non-inverted input */(b)
                            /* INT_CHAN_0=PC0=LOW -->             */
                            /* INT_CHAN_0 generate a high_pulse   */
     now_int_state=0;       /* now PC0=LOW                        */
     }
   else now_int_state=1;    /* now PC0=HIGH                       */
                            /* don't have to generate high_pulse  */
   }
else                        /* now PC0 is changed to HIGH         */(c)
   {                        /* --> INT_CHAN_0=PC0=HIGH now        */
   COUNT_H++;               /* find a HIGH_pulse (PC0)            */
   If((inport(wBase+7)&1)==1)/* the PC0 is still fixed in HIGH    */
     {                      /* need to generate a high_pulse      */
     outportb(wBase+0x2a,0);/* INV0 select the inverted input   */(d)
                            /* INT_CHAN_0=!PC0=LOW -->            */
                            /* INT_CHAN_0 generate a high_pulse   */
     now_int_state=1;       /* now PC0=HIGH                       */
     }
   else now_int_state=0;    /* now PC0=LOW                        */
                            /* don't have to generate high_pulse  */
   }
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```



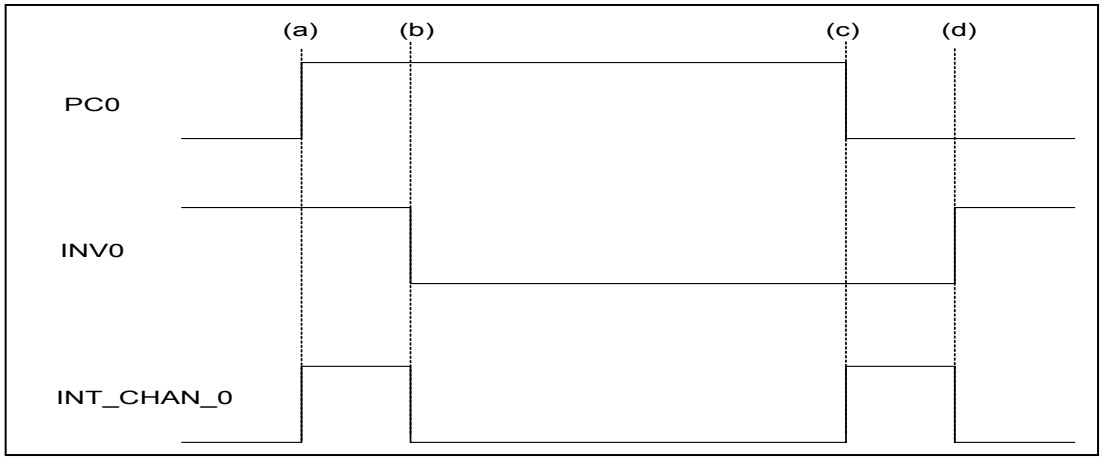
2.4.4 Initial_low, active_high Interrupt source

If the PC0 is a initial_low, active_high signal, the interrupt service routine should use INV0 to inverted/non-inverted the PC0 for high_pulse generation as follows: (Refer to DEMO3.C)

Initial setting:

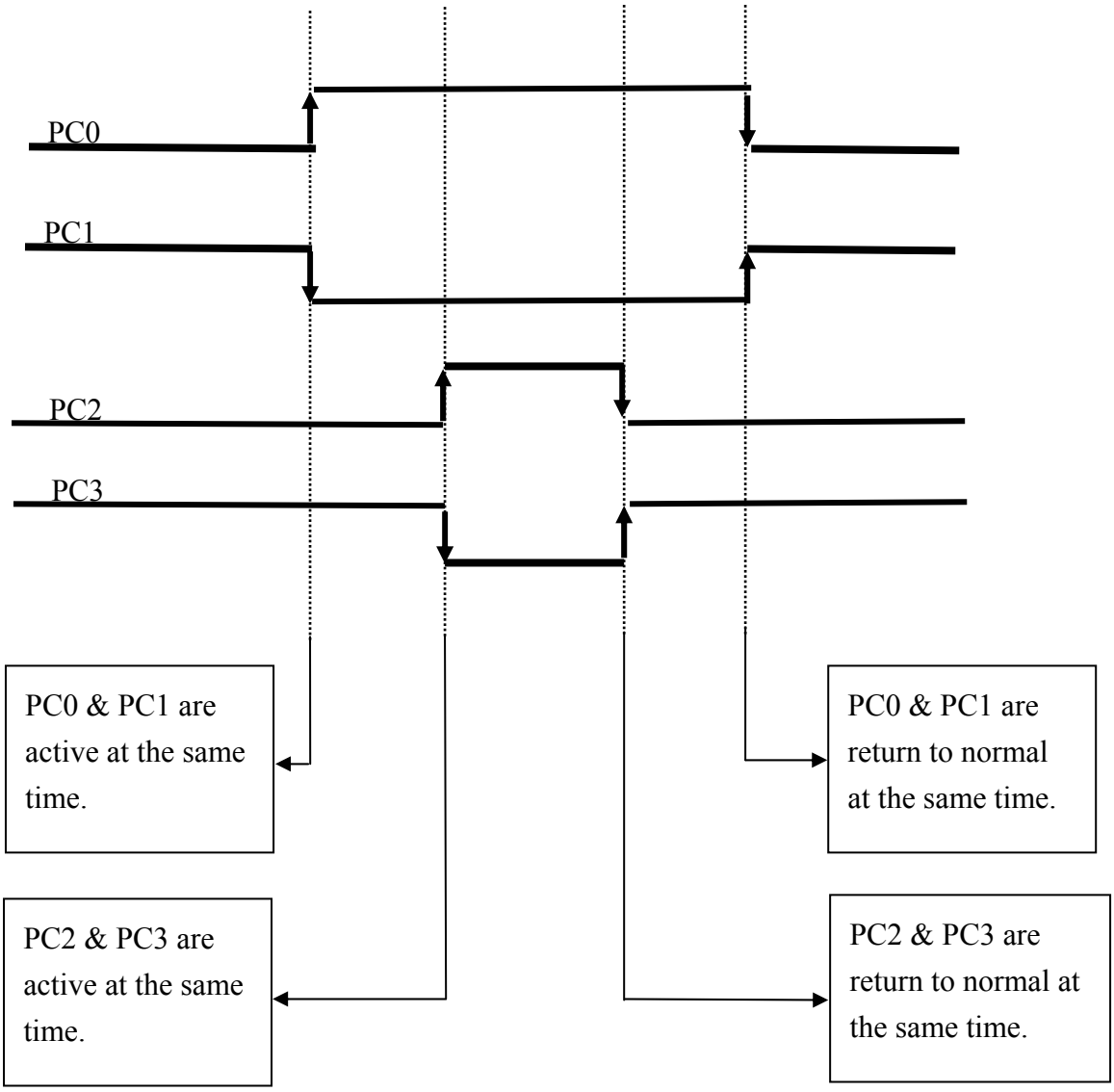
```
now_int_state=0;            /* initial state for PC0          */
outportb(wBase+0x2a,1);  /* select the non-inverted PC0 */
```

```
void interrupt irq_service()
{
if (now_int_state==1)       /* now PC0 is changed to LOW          */(c)
   {                        /* --> INT_CHAN_0=!PC0=HIGH now       */
   COUNT_L++;               /* find a LOW_pulse (PC0)             */
   If((inport(wBase+7)&1)==0)/* the PC0 is still fixed in LOW     */
     {                      /* → need to generate a high_pulse   */
     outportb(wBase+0x2a,1);/* INV0 select the non-inverted input */(d)
                            /* INT_CHAN_0=PC0=LOW -->             */
                            /* INT_CHAN_0 generate a high_pulse   */
     now_int_state=0;       /* now PC0=LOW                        */
     }
   else now_int_state=1;    /* now PC0=HIGH                       */
                            /* don't have to generate high_pulse  */
   }
else                        /* now PC0 is changed to HIGH         */(a)
   {                        /* --> INT_CHAN_0=PC0=HIGH now        */
   COUNT_H++;               /* find a High_pulse (PC0)            */
   If((inport(wBase+7)&1)==1)/* the PC0 is still fixed in HIGH    */
     {                      /* need to generate a high_pulse      */
     outportb(wBase+0x2a,0);/* INV0 select the inverted input   */(b)
                            /* INT_CHAN_0=!PC0=LOW -->            */
                            /* INT_CHAN_0 generate a high_pulse   */
     now_int_state=1;       /* now PC0=HIGH                       */
     }
   else now_int_state=0;    /* now PC0=LOW                        */
                            /* don't have to generate high_pulse  */
   }
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```



2.4.5 Muliti-Interrupt Source

Assume: PC0 is initial Low, active High,
PC1 is initial High, active Low
PC2 is initial Low, active High
PC3 is initial High, active Low
as follows :



Refer to DEMO5.C for source program. **All these four falling-edge & rising-edge can be detected by DEMO5.C.**

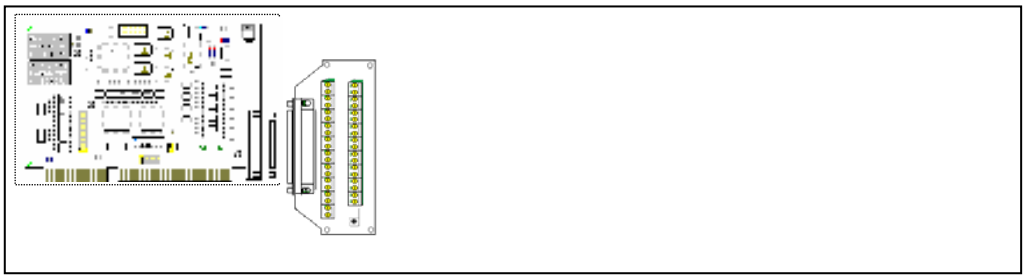
Note: When the interrupt is active, the user program has to identify the active signals. These signals may be active at the same time. The interrupt service routine has to service all active signals at the same time.

```
void interrupt irq_service()
{
new_int_state=inportb(wBase+7)&0x0f;  /* read all interrupt state */
int_c=new_int_state^now_int_state;    /* compare which interrupt  */
                                      /* signal be change         */
if ((int_c&0x1)!=0)                   /* INT_CHAN_0 is active     */
   {
   if ((new_int_state&0x01)!=0)  /* now PC0 change to high        */
      {
      CNT_H1++;
      }
   else                          /* now PC0 change to low         */
      {
      CNT_L1++;
      }
   invert=invert^1;              /* to generate a high pulse      */
   }

if ((int_c&0x2)!=0)
   {
   if ((new_int_state&0x02)!=0)  /* now PC1 change to high        */
      {
      CNT_H2++;
      }
   else                          /* now PC1 change to low         */
      {
      CNT_L2++;
      {
   invert=invert^2;              /* to generate a high pulse      */
   }

if ((int_c&0x4)!=0)
   {
   If ((new_int_state&0x04)!=0)  /* now PC2 change to high        */
      {
      CNT_H3++;
      }
   else                          /* now PC2 change to low         */
      {
      CNT_L3++;
      }
   invert=invert^4;              /* to generate a high pulse      */
   }

if ((int_c&0x8)!=0)
   {
   if ((new_int_state&0x08)!=0)  /* now PC3 change to high        */
      {
      CNT_H4++;
      {
   else                          /* now PC3 change to low         */
      {
      CNT_L4++;
      }
   invert=invert^8;              /* to generate a high pulse      */
   }
now_int_state=new_int_state;
outportb(wBase+0x2a,invert);
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```

2.5 Daughter Boards

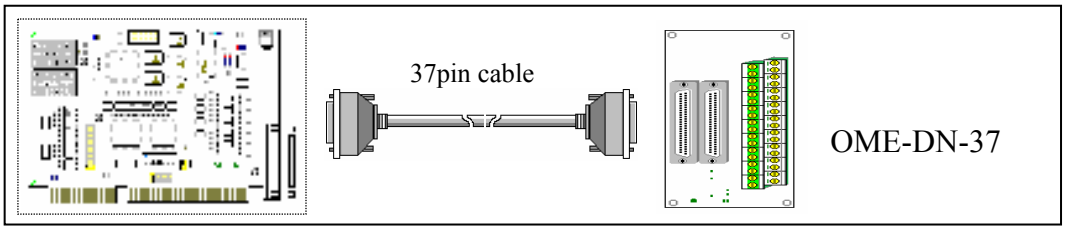
2.5.1 OME-DB-37

The OME-DB-37 is a general purpose daughter board for D-sub 37 pins, designed for an easy-wiring connection.



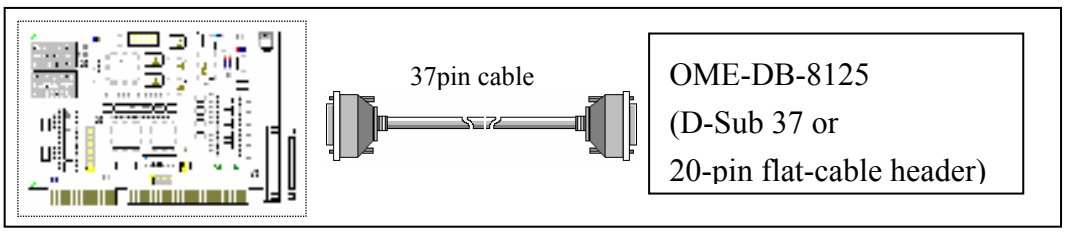
2.5.2 OME-DN-37

The OME-DN-37 is a general purpose daughter board for OME-DB-37 with DIN-Rail Mounting. It is designed for easy-wiring connection..



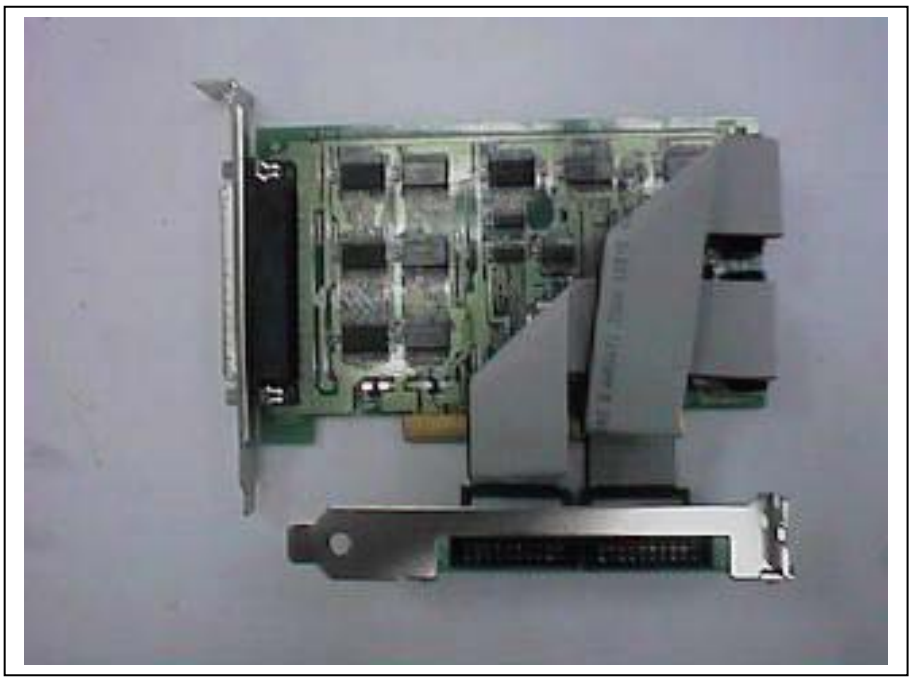
2.5.3 OME-DB-8125

The OME-DB-8125 is a general purpose screw terminal board. It is designed for easy wire connection. There is one D-Sub37 & two 20-pin flat-cable headers on the OME-DB-8125.



2.5.4 OME-ADP-20/PCI

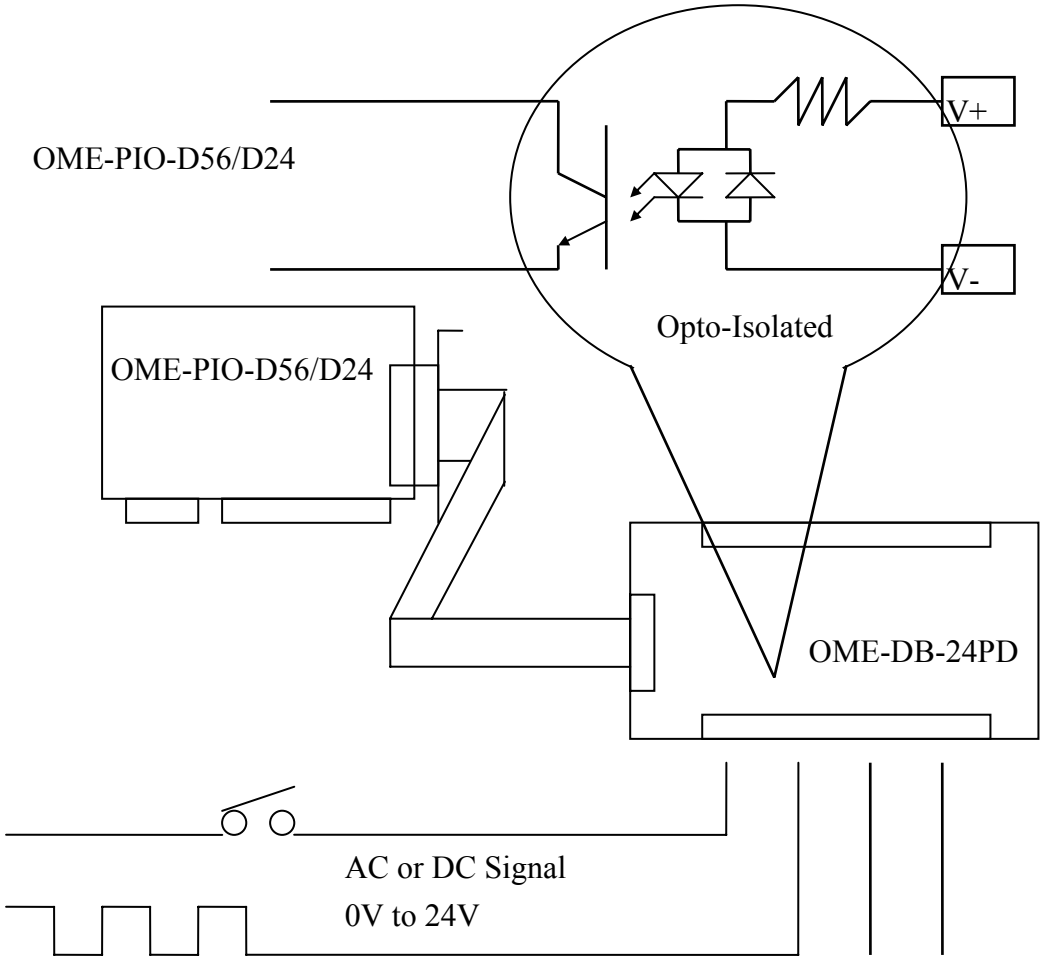
The OME-ADP-20/PCI is an extender for 20-pin header. One side of OME-ADP-20/PCI connects to a 20-pin header. The other side mounts on the PC chassis as follows:



NOTE: Please choose the suitable extender for your application

2.5.5 OME-DB-24PD Isolated Input Board

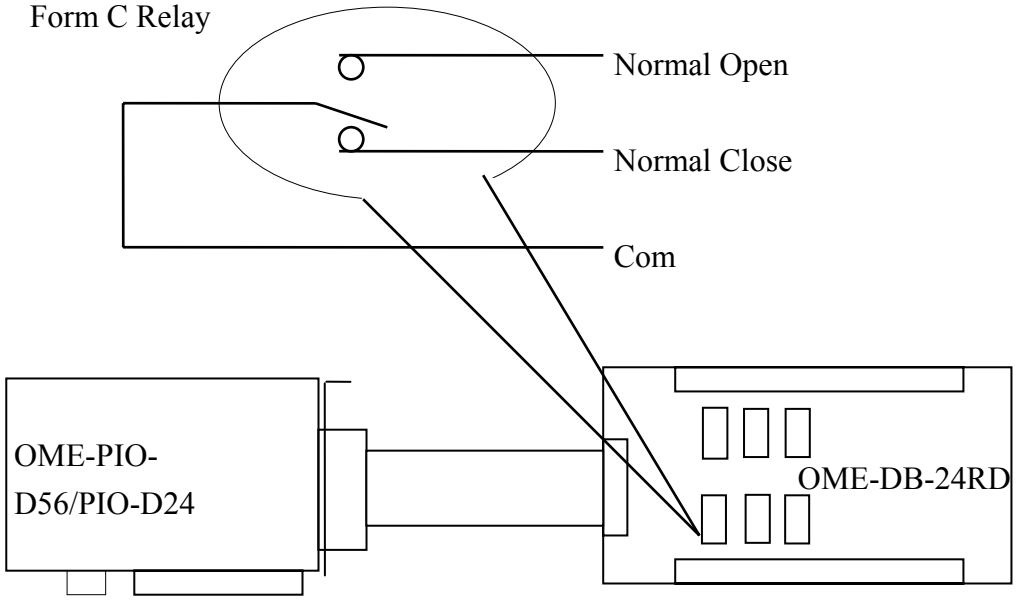
The OME-DB-24PD is a 24 channel isolated digital input daughter board. The optically isolated inputs of the OME-DB-24PD consist of a bi-directional opto-coupler with a resistor for current sensing. Use the OME-DB-24PD to sense DC signals from TTL levels up to 24V or use the OME-DB-24PD to sense a wide range of AC signals. Use this board to isolate the computer from large common-mode voltages, ground loops and transient voltage spikes that often occur in industrial environments.



	OME-DB-24PD
50-pin flat-cable header	Yes
D-sub 37-pin header	Yes

2.5.6 OME-DB-24RD Relay Board

The OME-DB-24RD, a 24 channel relay output board, consists of 24 form C relays for efficient, programmable load switching. The relays are energized by applying a 12V/24V signal to the appropriate relay channel on the 50-pin flat connector. There are 24 enunciator LEDs for each relay which light when their associated relay is activated.



Note:
Channel : 24 Form C Relay
Relay: Switch up to 0.5A at 110ACV
or 1A at 24DCV

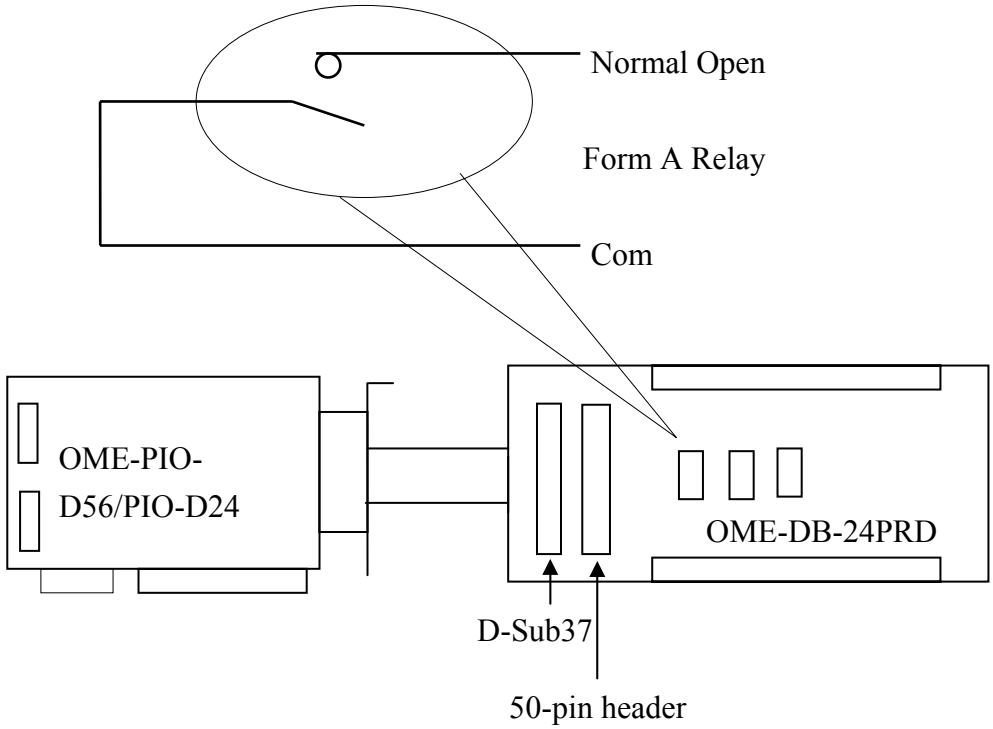
	OME-DB-24RD
50-pin flat-cable header	Yes
D-sub 37-pin header	Yes

OME-DB-24R, OME-DB-24RD	24*Relay (120V, 0.5A)
OME-DB-24PR,OME-DB-24PRD	24* Power Relay (250V, 5A)
OME-DB-24POR	24*photo MOS Relay (350V, 01.A)
OME-DB-24SSR	24*SSR (250VAC, 4A)
OME-DB-24C	24*O.C. (30V, 100 mA)
OME-DB-16P8R	16*Relay (120V, 0.5A) + 8*isolated input

2.5.7 OME-DB-24PRD, OME-DB-24POR, OME-DB-24C

OME-DB-24PRD	24*power relay, 5A/250V
OME-DB-24POR	24*photo MOS relay, 0.1A/350VAC
OME-DB-24C	24*open collector, 100mA per channel, 30V max.

The OME-DB-24PRD, a 24-channel power relay output board, consists of 8 form C and 16 form A electromechanical relays for efficient, programmable load control. The contact of each relay can control a 5A load at 250ACV/30VDCV. The relay is energized by applying a 5 volt signal to the appropriate relay channel on the 20-pin flat cable connector(using only 16 relays) or 50-pin flat cable connector(OPTO-22 compatible for DIO-24 series). Twenty four enunciator LEDs (one for each relay) light when their associated relay is activated. To avoid overloading your PC's power supply, this board needs a +12VDC or +24VDC external power supply.



Note:

50-Pin connector (OPTO-22 compatible), for OME-DIO-24/48/44, OME-PIO-D144/D96/D56/D48/D24

Channel: 16 Form A Relay, 8 Form C Relay

Relay: switches up to 5A at 110ACV / 5A at 30DCV

2.5.8 Daughter Board Comparison Table

	20-pin flat-cable	50-pin flat-cable	D-sub 37-pin
OME-DB-37	No	No	Yes
OME-DN-37	No	No	Yes
OME-ADP-37/PCI	No	Yes	Yes
OME-ADP-50/PCI	No	Yes	No
OME-DB-24P	No	Yes	No
OME-DB-24PD	No	Yes	Yes
OME-DB-16P8R	No	Yes	Yes
OME-DB-24R	No	Yes	No
OME-DB-24RD	No	Yes	Yes
OME-DB-24C	Yes	Yes	Yes
OME-DB-24PR	Yes	Yes	No
OME-DB-24PRD	No	Yes	Yes
OME-DB-24POR	Yes	Yes	Yes
OME-DB-24SSR	No	Yes	Yes

Note: There is no 50-pin flat cable header on OME-PIO-D56/OME-PIO-D24. The OME-PIO-D56/OME-PIO-D24 has one D-Sub 37 connector and two 20 pin flat-cable headers (only for OME-PIO-D56).

2.6 Pin Assignment

CON3: 37 pin of D-type female connector.

Pin Number	Description	Pin Number	Description
1	N.C.	20	VCC
2	N.C.	21	GND
3	P1B7	22	P2C7
4	P1B6	23	P2C6
5	P1B5	24	P2C5
6	P1B4	25	P2C4
7	P1B3	26	P2C3
8	P1B2	27	P2C2
9	P1B1	28	P2C1
10	P1B0	29	P2C0
11	GND	30	P0A7
12	N.C.	31	P0A6
13	GND	32	P0A5
14	N.C.	33	P0A4
15	GND	34	P0A3
16	N.C.	35	P0A2
17	GND	36	P0A1
18	VCC	37	P0A0
19	GND	XXXXXXX	This pin not available

All signals are TTL compatible.

CON2 : 20-pin header (only for OME-PIO-D56)

Pin Number	Description	Pin Number	Description
1	DI0	2	DI1
3	DI2	4	DI3
5	DI4	6	DI5
7	DI6	8	DI7
9	DI8	10	DI9
11	DI10	12	DI11
13	DI12	14	DI13
15	DI14	16	DI15
17	GND	18	GND
19	Vcc	20	+12V

CON1 : 20-pin header (only for OME-PIO-D56)

Pin Number	Description	Pin Number	Description
1	DO0	2	DO1
3	DO2	4	DO3
5	DO4	6	DO5
7	DO6	8	DO7
9	DO8	10	DO9
11	DO10	12	DO11
13	DO12	14	DO13
15	DO14	16	DO15
17	GND	18	GND
19	Vcc	20	+12V

All signals are TTL compatible.

3. I/O Control Register

3.1 How to Find the I/O Address

The plug & play BIOS will assign a proper I/O address to every OME-PIO/PISO series card in the power-up stage. The IDs of the cards are given below:

OME-PIO-D24

< REV 1.0 ~ REV 5.0 > :	< REV 6.0 or above > :
• Vendor ID = 0xE159	• Vendor ID = 0xE159
• Device ID = 0x0002	• Device ID = 0x0001
• Sub-vendor ID = 0x80	• Sub-vendor ID = 0xC080
• Sub-device ID = 0x01	• Sub-device ID = 0x01
• Sub-aux ID = 0x40	• Sub-aux ID = 0x40

OME-PIO-D56

< REV 1.0 ~ REV 4.0 > :	< REV 5.0 or above > :
• Vendor ID = 0xE159	• Vendor ID = 0xE159
• Device ID = 0x0002	• Device ID = 0x0001
• Sub-vendor ID = 0x80	• Sub-vendor ID = 0xC080
• Sub-device ID = 0x01	• Sub-device ID = 0x01
• Sub-aux ID = 0x40	• Sub-aux ID = 0x40

We provide all the following necessary functions:

1. **PIO_DriverInit(&wBoard, wSubVendor, wSubDevice, wSubAux)**
2. **PIO_GetConfigAddressSpace(wBoardNo,*wBase,*wIrq, *wSubVendor, *wSubDevice, *wSubAux, *wSlotBus, *wSlotDevice)**
3. **Show_PIO_PISO(wSubVendor, wSubDevice, wSubAux)**

All functions are defined in PIO.H. Refer to Chapter 4 for more information. The important driver information is given as follows:

1. Resource-allocated information:

- wBase : BASE address mapping in this PC
- wIrq: IRQ channel number allocated in this PC

2. PIO/PISO identification information:

- wSubVendor: subVendor ID of this board
- wSubDevice: subDevice ID of this board
- wSubAux: subAux ID of this board

3. PC's physical slot information:

- wSlotBus: hardware slot ID1 in this PC's slot position
- wSlotDevice: hardware slot ID2 in this PC's slot position

The utility program, **PIO_PISO.EXE**, will detect & show all OME-PIO/PISO cards installed in this PC. Refer to Sec. 4.1 for more information.

3.1.1 PIO_DriverInit

PIO_DriverInit(&wBoards, wSubVendor,wSubDevice,wSubAux)

- wBoards=0 to N → number of boards found in this PC
- wSubVendor → subVendor ID of board to find
- wSubDevice → subDevice ID of board to find
- wSubAux → subAux ID of board to find

This function can detect all OME-PIO/PISO series card in the system. It is implemented based on the PCI plug & play mechanism. It will find all OME-PIO/PISO series cards installed in this system & save all their resource information in the library.

Sample program 1: find all OME-PIO-D56/OME-PIO-D24 in the PC

```
wSubVendor=0x80; wSubDevice=1; wSubAux=0x40;  /* for OME-PIO-D56/D24 */
wRetVal=PIO_DriverInit(&wBoards, wSubVendor,wSubDevice,wSubAux);
printf("Threr are %d OME-PIO-D56/OME-PIO-D24 Cards in this PC\n",wBoards);

/* step2: save resources of all OME-PIO-D56/OME-PIO-D24 cards installed in this PC */
for (i=0; i<wBoards; i++)
  {
  PIO_GetConfigAddressSpace(i,&wBase,&wIrq,&wID1,&wID2,&wID3,
                            &wID4,&wID5);
  printf("\nCard_%d: wBase=%x, wIrq=%x", i,wBase,wIrq);
  wConfigSpace[i][0]=wBaseAddress;      /* save all resource of this card      */
  wConfigSpace[i][1]=wIrq;              /* save all resource of this card      */
  }
```

Sample program 2: find all OME-PIO/PISO in this PC (refer to Sec. 4.1 for more information)

```
wRetVal=PIO_DriverInit(&wBoards,0xff,0xff,0xff);  /*find all PIO_PISO*/
printf("\nThrer are %d PIO_PISO Cards in this PC",wBoards);
if (wBoards==0 ) exit(0);
printf("\n-----------------------------------------------------");
for(i=0; i<wBoards; i++)
   {
   PIO_GetConfigAddressSpace(i,&wBase,&wIrq,&wSubVendor,
               &wSubDevice,&wSubAux,&wSlotBus,&wSlotDevice);
   printf("\nCard_%d:wBase=%x,wIrq=%x,subID=[%x,%x,%x],
               SlotID=[%x,%x]",i,wBase,wIrq,wSubVendor,wSubDevice,
               wSubAux,wSlotBus,wSlotDevice);
   printf(" --> ");
   ShowPioPiso(wSubVendor,wSubDevice,wSubAux);
   }
```

The sub-IDs of OME-PIO/PISO series card are given as following:

OME-PIO/PISO series card	Description	Sub_vendor	Sub_device	Sub_AUX
OME-PIO-D144 (Rev 4.0)	144 * D/I/O	5C80	01	00
OME-PIO-D96 (Rev 4.0)	96 * D/I/O	5880	01	10
OME-PIO-D64 (Rev 2.0)	64 * D/I/O	4080	01	20
OME-PIO-D56 (Rev 6.0)	24* D/I/O + 16*D/I + 16*D/O	C080	01	40
OME-PIO-D48 (Rev 2.0)	48*D/I/O	0080	01	30
OME-PIO-D24 (Rev 6.0)	24*D/I/O	C080	01	40
OME-PIO-821	Multi-function	80	03	10
OME-PIO-DA16 (Rev 4.0)	16*D/A	4180	00	00
OME-PIO-DA8 (Rev 4.0)	8*D/A	4180	00	00
OME-PIO-DA4 (Rev 4.0)	4*D/A	4180	00	00
OME-PISO-C64 (Rev 4.0)	64 * isolated D/O (Current Sinking)	0280	00	00
OME-PISO-A64 (Rev 3.0)	64 * isolated D/O (Current Sourcing)	0280	00	50
OME-PISO-P64 (Rev 4.0)	64 * isolated D/I	0280	00	10
OME-PISO-P32C32 (Rev 5.0)	32 * isolated D/O (Current Sinking) +32 * isolated D/I	80	08	20
OME-PISO-P32A32 (Rev 3.0)	32 * isolated D/O (Current Sourcing) +32 * isolated D/I	8280	00	70
OME-PISO-P8R8 (Rev 2.0)	8* isolated D/I + 8 * 220V relay	4200	00	30
OME-PISO-P8SSR8AC (Rev 2.0)	8* isolated D/I + 8 * SSR /AC	4200	00	30
OME-PISO-P8SSR8DC (Rev 2.0)	8* isolated D/I + 8 * SSR /DC	4200	00	30
OME-PISO-730 (Rev 2.0)	16*DI + 16*D/O + 16* isolated D/I + 16* isolated D/O (Current Sinking)	C2FF	00	40
OME-PISO-730A (Rev 3.0)	16*DI + 16*D/O + 16* isolated D/I + 16* isolated D/O (Current Sourcing)	62FF	00	80
OME-PISO-813 (Rev 2.0)	32 * isolated A/D	4280	02	00
OME-PISO-DA2 (Rev 5.0)	2 * isolated D/A	4280	03	00

Note: If your board is a different version, it may also have different sub IDs. We offer the same function calls irrespective of the board version.

3.1.2 PIO_GetConfigAddressSpace

PIO_GetConfigAddressSpace(wBoardNo,*wBase,*wIrq, *wSubVendor,wSubDevice,*wSubAux,*wSlotBus, *wSlotDevice)**

- wBoardNo=0 to N → totally N+1 boards found by PIO_DriveInit(….)
- wBase → base address of the board control word
- wIrq → allocated IRQ channel number of this board
- wSubVendor → subVendor ID of this board
- wSubDevice → subDevice ID of this board
- wSubAux → subAux ID of this board
- wSlotBus → hardware slot ID1 of this board
- wSlotDevice → hardware slot ID2 of this board

The user can use this function to save resource of all OME-PIO/PISO cards installed in this system. Then the application program can control all functions of OME-PIO/PISO series card directly. The sample program is given as follows:

```
/* step1: detect all OME-PIO-D56/OME-PIO-D24 cards first */
wSubVendor=0x80; wSubDevice=1; wSubAux=0x40;  /*  for  OME-PIO-D56/D24 */
wRetVal=PIO_DriverInit(&wBoards, wSubVendor,wSubDevice,wSubAux);
printf("There are %d OME-PIO-D56/OME-PIO-D24 Cards in this PC\n",wBoards);

/* step2: save resource of all OME-PIO-D56/OME-PIO-D24 cards installed in this PC */
for (i=0; i<wBoards; i++)
  {
  PIO_GetConfigAddressSpace(i,&wBase,&wIrq,&t1,&t2,&t3,&t4,&t5);
  printf("\nCard_%d: wBase=%x, wIrq=%x", i,wBase,wIrq);
 wConfigSpace[i][0]=wBaseAddress; /* save all resource of this card          */
 wConfigSpace[i][1]=wIrq;         /* save all resource of this card          */
 }

/* step3: control the OME-PIO-D56/OME-PIO-D24 directly */
wBase=wConfigSpace[0][0];/* get base address the card_0                      */
outport(wBase,1);        /* enable all D/I/O operations of card_0            */

wBase=wConfigSpace[1][0];/* get base address the card_1                      */
outport(wBase,1);        /* enable all D/I/O operations of card_1            */
```

3.1.3 Show_PIO_PISO

Show_PIO_PISO(wSubVendor,wSubDevice,wSubAux)

- wSubVendor → subVendor ID of board to find
- wSubDevice → subDevice ID of board to find
- wSubAux → subAux ID of board to find

This function will output a text string for this special subIDs. This text string is the same as that defined in PIO.H

The demo program is given as follows:

```
wRetVal=PIO_DriverInit(&wBoards,0xff,0xff,0xff);  /*find all PIO_PISO*/
printf("\nThrer are %d PIO_PISO Cards in this PC",wBoards);
if (wBoards==0 ) exit(0);

printf("\n-----------------------------------------------------");
for(i=0; i<wBoards; i++)
   {
   PIO_GetConfigAddressSpace(i,&wBase,&wIrq,&wSubVendor,
               &wSubDevice,&wSubAux,&wSlotBus,&wSlotDevice);

   printf("\nCard_%d:wBase=%x,wIrq=%x,subID=[%x,%x,%x],
               SlotID=[%x,%x]",i,wBase,wIrq,wSubVendor,wSubDevice,
               wSubAux,wSlotBus,wSlotDevice);
   printf(" --> ");
   ShowPioPiso(wSubVendor,wSubDevice,wSubAux);
   }
```

3.2 The Assignment of I/O Address

The plug & play BIOS will assign the proper I/O address to the OME-PIO/PISO series card. If there is only one OME-PIO/PISO board, the user can identify the board as card_0. If there are two PIO/PISO boards in the system, the user will be very difficult to identify which board is card_0? The software driver can support 16 boards max. Therefore the user can install 16 boards of PIO/PSIO series in one PC system. How to find the card_0 & card_1?

The simplest way to identify which card is card_0 is to use wSlotBus & wSlotDevice as follows :

1. Remove all OME-PIO-D56/OME-PIO-D24 from this PC
2. Install one OME-PIO-D56/OME-PIO-D24 into the PC's PCI_slot1, run PIO_PISO.EXE & record the wSlotBus1 & wSlotDevice1
3. Remove all OME-PIO-D56/OME-PIO-D24 from this PC
4. Install one OME-PIO-D56/D24 into the PC's PCI_slot2, run PIO_PISO.EXE & record the wSlotBus2 & wSlotDevice2
5. Repeat (3) & (4) for all PCI_slot?, record all wSlotBus? & wSlotDevice?

The records may be as follows:

PC's PCI slot	**wSlotBus**	**wSlotDevice**
Slot_1	**0**	**0x07**
Slot_2	**0**	**0x08**
Slot_3	**0**	**0x09**
Slot_4	**0**	**0x0A**
PCI-BRIDGE		
Slot_5	**1**	**0x0A**
Slot_6	**1**	**0x08**
Slot_7	**1**	**0x09**
Slot_8	**1**	**0x07**

The above procedure will record all wSlotBus? & wSlotDevice? in this PC. These values will be mapped to this PC's physical slot. This mapping will not be changed for any PIO/PISO cards. So it can be used to identify the specified OME-PIO/PISO card as follows:

Step 1: Record all wSlotBus? & wSlotDevice?

Step2: Use PIO_GetConfigAddressSpace(…) to get the specified card's wSlotBus & wSlotDevice

Step3: The user can identify the specified OME-PIO/PISO card if the compare the wSlotBus & wSlotDevice in step2 to step1.

3.3 The I/O Address Map

The I/O address of OME-PIO/PISO series card is automatically assigned by the main board ROM BIOS. The I/O address can also be re-assigned by user. **Users are strongly recommended to change the auto-assigned I/O address. The plug & play BIOS will assign proper I/O address to each OME-PIO/PISO series card very well.** The I/O addresses of OME-PIO-D56/OME-PIO-D24 are given as follows:

Address	**Read**	**Write**
Wbase+0	RESET\ control register	Same
Wbase+2	Aux control register	Same
Wbase+3	Aux data register	Same
Wbase+5	INT mask control register	Same
Wbase+7	Aux pin status register	Same
Wbase+0x2a	INT polarity control register	Same
Wbase+0xc0	read Port0	write Port0
Wbase+0xc4	read Port1	write Port1
Wbase+0xc8	read Port2	write Port2
Wbase+0xcc	read configuration	Port0~Port2 configuration
Wbase+0xd0	read CON2 Low byte (only for OME-PIO-D56)	write CON1 Low byte (only for OME-PIO-D56)
Wbase+0xd4	read CON2 high byte (only for OME-PIO-D56)	write CON1 high byte (only for OME-PIO-D56)

Note. Refer to Sec. 3.1 for more information about wBase.

3.3.1 RESET\ Control Register

(Read/Write): wBase+0

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	RESET\

Note. Refer to Sec. 3.1 for more information about wBase.

When the PC is first powered up, the RESET\ signal is in Low-state. **This will disable all D/I/O operations.** The user has to set the RESET\ signal to High-state before any D/I/O command.

```
outportb(wBase,1);      /*  RESET\=High → all D/I/O are enable now */
outportb(wBase,0);      /*  RESET\=Low → all D/I/O are disable now */
```

3.3.2 AUX Control Register

(Read/Write): wBase+2

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Aux7	Aux6	Aux5	Aux4	Aux3	Aux2	Aux1	Aux0

Note. Refer to Sec. 3.1 for more information about wBase.

Aux?=0→ this Aux is used as a D/I

Aux?=1→ this Aux is used as a D/O

When the PC is first powered up, All Aux? signals are in Low-state. All Aux? are designed as D/I for all OME-PIO/PISO series. Please set all Aux? in D/I state.

3.3.3 AUX data Register

(Read/Write): wBase+3

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Aux7	Aux6	Aux5	Aux4	Aux3	Aux2	Aux1	Aux0

Note. Refer to Sec. 3.1 for more information about wBase.

When the Aux? is used as D/O, the output state is controlled by this register. This register is designed and reserved for feature extension, so do not control this register now.

3.3.4 INT Mask Control Register

(Read/Write): wBase+5

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	EN3	EN2	EN1	EN0

Note. Refer to Sec. 3.1 for more information about wBase.

EN0=0→ disable PC0 as a interrupt signal (default)

EN0=1→ enable PC0 as a interrupt signal

```
outportb(wBase+5,0);        /* disable interrupt                    */
outportb(wBase+5,1);        /* enable interrupt PC0                 */
outportb(wBase+5,0x0f);     /* enable interrupt PC0, PC1,PC2,PC3    */
```

3.3.5 Aux Status Register

(Read/Write): wBase+7

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Aux7	Aux6	Aux5	Aux4	Aux3	Aux2	Aux1	Aux0

Note. Refer to Sec. 3.1 for more information about wBase.

Aux0=PC0, Aux1=PC1, Aux2=PC2, Aux3=PC3, Aux7~4=Aux-ID. Refer to DEMO5.C for more information. The Aux 0~3 are used as interrupt source. The interrupt service routine has to read this register for interrupt source identification. Refer to Sec. 2.5 for more information.

3.3.6 Interrupt Polarity Control Register

(Read/Write): wBase+0x2A

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	INV3	INV2	INV1	INV0

Note. Refer to Sec. 3.1 for more information about wBase.

INV0=1→ select the non-inverted signal from PC0
INV0=0→ select the inverted signal from PC0

```
outportb(wBase+0x2a,0x0f); /* select the non-inverted input PC0/1/2/3      */
outportb(wBase+0x2a,0x00); /* select the inverted input of PC0/1/2/3       */

outportb(wBase+0x2a,0x0e); /* select the inverted input of PC0             */
                           /* select the non-inverted input PC1/2/3        */

outportb(wBase+0x2a,0x0c); /* select the inverted input of PC0/1           */
                           /* select the non-inverted input PC2/3          */
```

Refer to Sec. 2.4 for more information.
Refer to DEMO5.C for more information.

3.3.7 I/O Selection Control Register

(Write): wBase+0xcc

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	Port2	Port1	Port0

Note. Refer to Sec. 3.1 for more information about wBase.

Port? = 1→ this port is used as a D/O port
Port? = 0→ this port is used as a D/I port

```
outportb(wBase+0xcc,0x00);     /* configure Port0/1/2 as D/I port          */

outportb(wBase+0xcc,0x04);     /* configure Port0/1 as D/I port            */
                               /* configure Port2 as D/O port              */
```

3.3.8 Read/Write 8-bit data Register

(Read/Write):wBase+0xc0/0xc4/0xc8/0xd0/0xd4

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
D7	D6	D5	D4	D3	D2	D1	D0

Note. Refer to Sec. 3.1 for more information about wBase.

There are five/three 8-bit I/O ports in the OME-PIO-D56/OME-PIO-D24. Each port is easy to read/write to by access to their own data registers.

```
outportb(wBase+0xc0,Val);          /* write to D/O port                  */
Val=inportb(wBase+0xc0);           /* read from D/I port                 */

outportb(wBase+0xcc,0x07);         /* configure Port0~Port2 as DO port   */
outportb(wBase+0xc0,i1);           /* write to Port0                     */
outportb(wBase+0xc0,i2);           /* write to Port1                     */
outportb(wBase+0xc0,i3);           /* write to Port2                     */

outportb(wBase+0xcc,0x01);         /* configure Port0 as DO port         */
                                   /* Port1~Port2 as DI port             */
outportb(wBase+0xc0,i1);           /* write to Port0                     */
j2=inportb(wBase+0xc4);            /* read Port1                         */
j3=inportb(wBase+0xc8);            /* read Port2                         */

l=inportb(wBase+0xd0);             /* read CON2 Low byte                 */
h=inportb(wBase+0xd4);             /* read CON2 High byte                */
Val=(h<<8)+l;                      /* Val is 16 bit data                 */

outportb(wBase+0xd0,Val);          /* write to CON1 Low byte             */
outportb(wBase+0xd4,(Val>>8));     /* write to CON1 high byte            */
```

4. Demo program

It is recommended to read the release notes first. Important information will be given in release note as follows:

1. Where you can find the software driver & utility?
2. **How to install software & utility?**
3. Where is the diagnostic program?
4. FAQs

There are many demo programs available on the software floppy disk or CD. After the software installation, the driver will be installed into disk as follows:

- \TC*.* → for Turbo C 2.xx or above
- \MSC*.* → for MSC 5.xx or above
- \BC*.* → for BC 3.xx or above

- \TC\LIB*.* → for TC library
- \TC\DEMO*.* → for TC demo program

- \TC\LIB\Large*.* → TC large model library
- \TC\LIB\Huge*.* → TC huge model library
- \TC\LIB\Large\PIO.H → TC declaration file
- \TC\\LIB\Large\TCPIO_L.LIB → TC large model library file
- \TC\LIB\Huge\PIO.H → TC declaration file
- \TC\\LIB\Huge\TCPIO_H.LIB → TC huge model library file

- \MSC\LIB\Large\PIO.H → MSC declaration file
- \MSC\LIB\Large\MSCPIO_L.LIB → MSC large model library file
- \MSC\LIB\Huge\PIO.H → MSC declaration file
- \MSC\\LIB\Huge\MSCPIO_H.LIB → MSC huge model library file

- \BC\LIB\Large\PIO.H → BC declaration file
- \BC\LIB\Large\BCPIO_L.LIB → BC large model library file
- \BC\LIB\Huge\PIO.H → BC declaration file
- \BC\\LIB\Huge\BCPIO_H.LIB → BC huge model library file

NOTE: The library is validated for all OME-PIO/PISO series cards.

4.1 PIO_PISO

```
/* ------------------------------------------------------------ */
/* Find all PIO_PISO series cards in this PC system             */
/* step 1 : plug all PIO_PISO cards into PC                     */
/* step 2 : run PIO_PISO.EXE                                    */
/* ------------------------------------------------------------ */

#include "PIO.H"

WORD wBase,wIrq;
WORD wBase2,wIrq2;

int main()
{
int  i,j,j1,j2,j3,j4,k,jj,dd,j11,j22,j33,j44;
WORD wBoards,wRetVal;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
char c;
float ok,err;

clrscr();
wRetVal=PIO_DriverInit(&wBoards,0xff,0xff,0xff);  /*for PIO-PISO*/
printf("\nThrer are %d PIO_PISO Cards in this PC",wBoards);
if (wBoards==0 ) exit(0);

printf("\n-------------------------------------------------------");
for(i=0; i<wBoards; i++)
   {
   PIO_GetConfigAddressSpace(i,&wBase,&wIrq,&wSubVendor,
               &wSubDevice,&wSubAux,&wSlotBus,&wSlotDevice);

   printf("\nCard_%d:wBase=%x,wIrq=%x,subID=[%x,%x,%x],
               SlotID=[%x,%x]",i,wBase,wIrq,wSubVendor,wSubDevice,
               wSubAux,wSlotBus,wSlotDevice);
   printf(" --> ");
   ShowPioPiso(wSubVendor,wSubDevice,wSubAux);
   }

PIO_DriverClose();
}
```

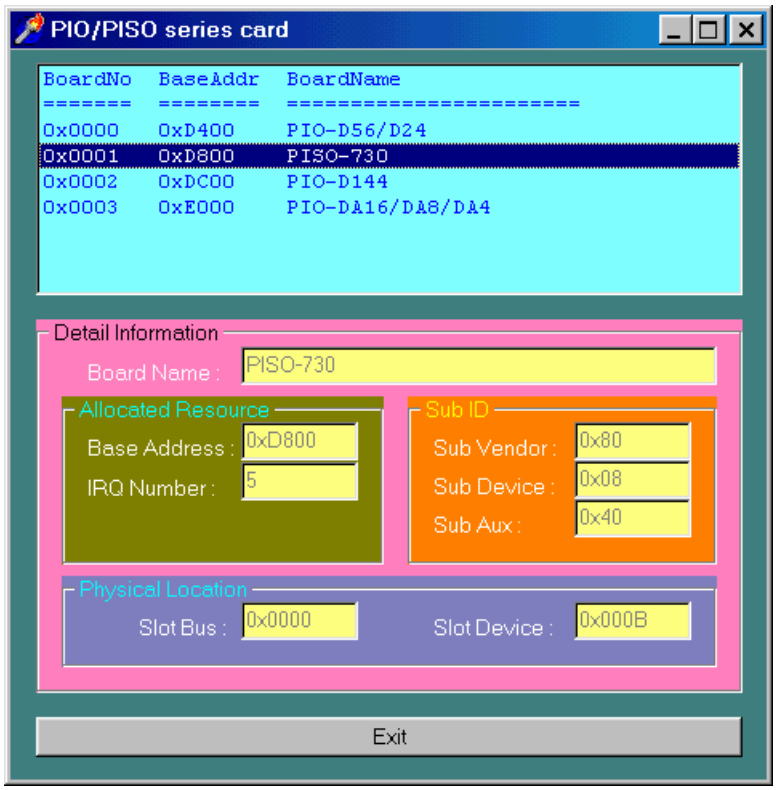
NOTE**: the PIO_PISO.EXE is valid for all PIO/PISO cards**. The user can execute the PIO_PISO.EXE to get the following information:

- List all PIO/PISO cards installed in this PC
- List all resources allocated to every PIO/PISO cards
- List the wSlotBus & wSlotDevice for specified PIO/PISO card identification. (Refer to Sec. 3.2 for more information)

4.1.1 PIO_PISO.EXE for Windows

User can find this utility in the company CD or floppy disk. It is useful for all OME-PIO/PISO series card.

After executing the utility, detailed information for all OME-PIO/PISO cards that installed in the PC will be show as follows:



4.2 DEMO1

```
/* demo 1 : D/O demo of CON3                                          */
/* step 1 : connect a OME-DB-24C to CON3 of OME-PIO-D56/D24           */
/* step 2 : run DEMO1.EXE                                             */
/* step 3 : LEDs of OME-DB-24C will turn on sequentially              */
/* ------------------------------------------------------------------ */
#include "PIO.H"

WORD wBase,wIrq;

int main()
{
int  i1,i2,i3;
long i;
WORD wBoards,wRetVal,t1,t2,t3,t4,t5,t6;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
char c;

clrscr();

/* step1 : find address-mapping of PIO/PISO cards                     */
wRetVal=PIO_DriverInit(&wBoards,0x80,0x01,0x40);/* for OME-PIO-
D56/D24*/
printf("\n(1) Threr are %d OME-PIO-D56/D24 Cards in this PC",wBoards);
if ( wBoards==0 ) exit(0);
printf("\n\n-------------- The Configuration Space --------------");
for(i=0;i<wBoards;i++)
   {
   PIO_GetConfigAddressSpace(i,&wBase,&wIrq,&wSubVendor,
   &wSubDevice,&wSubAux,&wSlotBus,&wSlotDevice);
   printf("\nCard_%d:wBase=%x,wIrq=%x,subID=[%x,%x,%x],SlotID=
   [%x,%x]",i,wBase,wIrq,wSubVendor,wSubDevice,
   wSubAux,wSlotBus,wSlotDevice);
   printf(" --> ");
   ShowPioPiso(wSubVendor,wSubDevice,wSubAux);
   }
PIO_GetConfigAddressSpace(0,&wBase,&wIrq,&t1,&t2,&t3,&t4,&t5);
                                          /* select card_0 */
/* step2 : enable all D/I/O port                                      */
outportb(wBase,1);                        /* /RESET -> 1   */
/* step3 : configure I/O direction                                    */
outportb(wBase+0xcc,0x07);              /* set CON3 as D/O ports */

i=1;
for (;;)
    {
    i1=i&0xff;
    i2=(i>>8)&0xff;
    i3=(i>>16)&0xff;
    outportb(wBase+0xc0,i1);
    outportb(wBase+0xc4,i2);
    outportb(wBase+0xc8,i3);
    delay(10000);
    i=i<<1;
    i=i&0x0ffffff;
    if (i==0) i=1;
    if (kbhit()!=0) break;
    }
PIO_DriverClose();
}
```

4.3 DEMO2

```
/* demo 2 : DI/O demo of CON1, CON2 & CON3                          */
/* step 1 : connect OME-DB-24P to CON3 of OME-PIO-D56/D24           */
/*        : connect CON1 t0 CON2 of OME-PIO-D56                     */
/* step 2 : run DEMO2.EXE                                           */
/* step 3 : check the information on screen D/I will same as D/O    */
/*        : check the result on screen will same as CON3 input      */
/* ---------------------------------------------------------------- */

#include "PIO.H"

WORD wBase,wIrq;

int main()
{
int  i1,i2,i3,j1,j2,j3;
WORD wBoards,wRetVal,t1,t2,t3,t4,t5,t6;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
char c;
long i;

clrscr();

/* step1 : find address-mapping of PIO/PISO cards                   */
           .
           .
/* step2 : enable all D/I/O port                                    */
outportb(wBase,1);                                 /* /RESET -> 1  */

/* step3 : configure I/O direction                                  */
outportb(wBase+0xcc,0x00);               /* set CON3 as D/I ports */
i=1;

for (;;)
    {
    gotoxy(1,7);
    i1=i&0xff;
    i2=(i>>8)&0xff;
    outportb(wBase+0xd0,i1);
    outportb(wBase+0xd4,i2);
    j1=inportb(wBase+0xd0);
    j2=inportb(wBase+0xd4);
    printf("\nDO = [%2x,%2x], DI = [%2x,%2x]",i2,i1,j2,j1);
    if ((j1!=i1)||(j2!=i2))
       {
       printf("\n\nError ......");
       }
    else printf("\nO.K.  ......");
    j1=inportb(wBase+0xc0);
    j2=inportb(wBase+0xc4);
    j3=inportb(wBase+0xc8);
    printf("\n\nD/I of CON3 [PA, PB, PC] = [%2x,%2x,%2x] ",j1,j2,j3);
    i=i<<1;
    i=i&0x0ffff;
    if (i==0) i=1;
    if (kbhit()!=0) return;
    }
PIO_DriverClose();
}
```

4.4 DEMO3

```
/* demo 3 : Count high pulse of PC0                                  */
/*          (initial Low & active High)                              */
/* step 1 : run DEMO3.EXE                                            */
/* ----------------------------------------------------------------- */

#include "PIO.H"

#define A1_8259 0x20
#define A2_8259 0xA0
#define EOI     0x20

WORD init_low();
WORD wBase,wIrq;

static void interrupt irq_service();
int COUNT,irqmask,now_int_state;

int main()
{
int  i,j;
WORD wBoards,wRetVal,t1,t2,t3,t4,t5,t6;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
char c;

clrscr();

/* step1 : find address-mapping of PIO/PISO cards                    */
           .
           .
                                                 /* select card_0 */
/* step2 : enable all D/I/O port                                     */
outportb(wBase,1);                                /* /RESET -> 1  */

/* step3 : configure I/O direction                                   */
outportb(wBase+0xcc,0x00);               /* set CON3 as D/I ports */

COUNT=0;
init_low();
printf("\n\n***** show the count of High_pulse *****\n");

for (;;)
    {
    gotoxy(1,8);
    printf("\nCOUNT=%d",COUNT);
    if (kbhit()!=0) break;
    }
outportb(wBase+5,0);                     /* disable all interrupt */

PIO_DriverClose();
}

/* Use PC0 as external interrupt signal                              */
WORD init_low()
{
disable();
outportb(wBase+5,0);                      /* disable all interrupt */
if (wIrq<8)
   {
```

```
   irqmask=inportb(A1_8259+1);
   outportb(A1_8259+1,irqmask & 0xff ^ (1<<wIrq));
   setvect(wIrq+8,irq_service);
   }
else
   {
   irqmask=inportb(A1_8259+1);
   outportb(A1_8259+1,irqmask & 0xfb);                    /* IRQ2 */
   irqmask=inportb(A2_8259+1);
   outportb(A2_8259+1,irqmask & 0xff ^ (1<<(wIrq-8)));
   setvect(wIrq-8+0x70,irq_service);
   }

outportb(wBase+5,1);                  /* enable interrupt (PC0)  */
now_int_state=0;                      /* now ini_signal is low   */
outportb(wBase+0x2a,1);               /* select the non-inverte  */
enable();
}

void interrupt irq_service()
{
if (now_int_state==1)             /* now PC0 change to low         */
   {                              /* INT_CHAN_0 = !PC0             */
   if ((inportb(wBase+7)&1)==0)   /* PC0 still fixed in low        */
      {                           /* need to generate a high pulse */
      outportb(wBase+0x2a,1);     /* INV0 select noninverted input */
      now_int_state=0;            /* now PC0=low                   */
      }
   else now_int_state=1;          /* now PC0=High                  */
   }
else                              /* now PC0 change to high        */
   {                              /* INT_CHAN_0 = PC0              */
   COUNT++;
   if ((inportb(wBase+7)&1)==1)   /* PC0 still fixed in high       */
      {                           /* need to generate a high pulse */
      outportb(wBase+0x2a,0);     /* INV0 select inverted input    */
      now_int_state=1;            /* now PC0=high                  */
      }
   else now_int_state=0;          /* now PC0=low                   */
   }

if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```

4.5 DEMO4

```
/* demo 4 : Count high pulse of PC0                                  */
/*          (initial High & active Low)                              */
/* step 1 : run DEMO4.EXE                                            */
/* ----------------------------------------------------------------- */

#include "PIO.H"

#define A1_8259 0x20
#define A2_8259 0xA0
#define EOI     0x20

WORD init_high();
WORD wBase,wIrq;

static void interrupt irq_service();
int COUNT,irqmask,now_int_state;

int main()
{
int  i,j;
WORD wBoards,wRetVal,t1,t2,t3,t4,t5,t6;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
char c;

clrscr();

/* step1 : find address-mapping of PIO/PISO cards                    */
           .
           .
                                                 /* select card_0 */
/* step2 : enable all D/I/O port                                     */
outportb(wBase,1);                                /* /RESET -> 1  */

/* step3 : configure I/O direction                                   */
outportb(wBase+0xcc,0x00);               /* set CON3 as D/I ports */

COUNT=0;
init_high();
printf("\n\n***** show the count of Low_pulse *****\n");

for (;;)
    {
    gotoxy(1,7);
    printf("\nCOUNT=%d",COUNT);
    if (kbhit()!=0) break;
    }
outportb(wBase+5,0);                     /* disable all interrupt */

PIO_DriverClose();
}

/* Use PC0 as external interrupt signal                              */
WORD init_high()
{
disable();
outportb(wBase+5,0);                     /* disable all interrupt */
if (wIrq<8)
   {
   irqmask=inportb(A1_8259+1);
   outportb(A1_8259+1,irqmask & 0xff ^ (1<<wIrq));
   setvect(wIrq+8,irq_service);
```

```
   }
else
   {
   irqmask=inportb(A1_8259+1);
   outportb(A1_8259+1,irqmask & 0xfb);                    /* IRQ2 */
   irqmask=inportb(A2_8259+1);
   outportb(A2_8259+1,irqmask & 0xff ^ (1<<(wIrq-8)));
   setvect(wIrq-8+0x70,irq_service);
   }

outportb(wBase+5,1);                  /* enable interrupt (PC0)  */
now_int_state=1;                      /* now ini_signal is high  */
outportb(wBase+0x2a,0);               /* select the inverte      */
enable();
}

void interrupt irq_service()
{
if (now_int_state==1)            /* now PC0 change to low         */
   {                             /* INT_CHAN_0 = !PC0             */
   COUNT++;
   if ((inportb(wBase+7)&1)==0)  /* PC0 still fixed in low        */
      {                          /* need to generate a high pulse */
      outportb(wBase+0x2a,1);    /* INV0 select noninverted input */
      now_int_state=0;           /* now PC0=low                   */
      }
   else now_int_state=1;         /* now PC0=High                  */
   }
else                             /* now PC0 change to high        */
   {                             /* INT_CHAN_0 = PC0              */
   if ((inportb(wBase+7)&1)==1)  /* PC0 still fixed in high       */
      {                          /* need to generate a high pulse */
      outportb(wBase+0x2a,0);    /* INV0 select inverted input    */
      now_int_state=1;           /* now PC0=high                  */
      }
   else now_int_state=0;         /* now PC0=low                   */
   }
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```

4.6 DEMO5

```
/* demo 5 : Four interrupt source                                    */
/*          PC0  : initial Low  , active High                        */
/*          PC1  : initial High , active Low                         */
/*          PC2  : initial Low  , active High                        */
/*          PC3  : initial High , active Low                         */
/* step 1 : run DEMO5.EXE                                            */
/* ----------------------------------------------------------------- */

#include "PIO.H"

#define A1_8259 0x20
#define A2_8259 0xA0
#define EOI     0x20

WORD init();
WORD wBase,wIrq;

static void interrupt irq_service();
int irqmask,now_int_state,new_int_state,invert,int_c,int_num;
int CNT_L1,CNT_L2,CNT_L3,CNT_L4;
int CNT_H1,CNT_H2,CNT_H3,CNT_H4;

int main()
{
int  i,j;
WORD wBoards,wRetVal,t1,t2,t3,t4,t5,t6;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
char c;

clrscr();

/* step1 : find address-mapping of PIO/PISO cards                    */
           .
           .
                                                  /* select card_0 */
/* step2 : enable all D/I/O port                                     */
outportb(wBase,1);                                 /* /RESET -> 1  */

/* step3 : configure I/O direction                                   */
outportb(wBase+0xcc,0x00);               /* set CON3 as D/I ports */

init();
printf("\n***** show the count of pulse *****\n");

for (;;)
    {
    gotoxy(1,7);
    printf("\n(CNT_L,CNT_H)=(%d,%d) (%d,%d) (%d,%d) (%d,%d)
%x",CNT_L1,CNT_H1,CNT_L2,CNT_H2,CNT_L3,CNT_H3,CNT_L4,CNT_H4,int_num);
    if (kbhit()!=0) break;
    }

outportb(wBase+5,0);                     /* disable all interrupt */
PIO_DriverClose();
}

/* Use PC0, PC1, PC2 & PC3 as external interrupt signal             */
WORD init()
{
disable();
outportb(wBase+5,0);                     /* disable all interrupt */
```

```
if (wIrq<8)
   {
   irqmask=inportb(A1_8259+1);
   outportb(A1_8259+1,irqmask & 0xff ^ (1<<wIrq));
   setvect(wIrq+8,irq_service);
   }
else
   {
   irqmask=inportb(A1_8259+1);
   outportb(A1_8259+1,irqmask & 0xfb);                    /* IRQ2 */
   irqmask=inportb(A2_8259+1);
   outportb(A2_8259+1,irqmask & 0xff ^ (1<<(wIrq-8)));
   setvect(wIrq-8+0x70,irq_service);
   }

invert=0x05;
outportb(wBase+0x2a,invert);          /* PC0   = non-inverte input */
                                      /* PC1   =     inverte input */
                                      /* PC2   = non-inverte input */
                                      /* PC3   =     inverte input */

now_int_state=0x0a;                   /* PC0   = Low               */
                                      /* PC1   = High              */
                                      /* PC2   = Low               */
                                      /* PC3   = High              */

CNT_L1=CNT_L2=CNT_L3=CNT_L4=0;        /* Low_pulse counter         */
CNT_H1=CNT_H2=CNT_H3=CNT_H4=0;        /* High_pulse counter        */
int_num=0;
outportb(wBase+5,0x0f);               /* enable interrupt PC0,PC1  */
enable();                             /* PC2,PC3                   */
}
/* ---------------------------------------------------------------- */
/* NOTE:1.The hold-time of INT_CHAN_0/1/2/3 must long enough      */
/*      2.The ISR must read the interrupt status again to the     */
/*        active interrupt sources.                               */
/*      3.The INT_CHAN_0&INT_CHAN_1 can be active at the same time*/
/* ---------------------------------------------------------------- */
void interrupt irq_service()
{
char c;

int_num++;
new_int_state=inportb(wBase+7)&0x0f;  /* read all interrupt state */
int_c=new_int_state^now_int_state;    /* compare which interrupt  */
                                      /* signal be change         */
if ((int_c&0x1)!=0)                   /* INT_CHAN_0 is active     */
   {
   if ((new_int_state&0x1)!=0)/* now PC0 is change to high        */
      {
      CNT_H1++;
      }
   else                       /* now PC0 is change to low         */
      {
      CNT_L1++;
      }
   invert=invert^1;           /* to generate a high pulse         */
   }
if ((int_c&0x2)!=0)                   /* INT_CHAN_1 is active     */
   {
   if ((new_int_state&0x2)!=0)/* now PC1 is change to high        */
      {
      CNT_H2++;
      }
```

```
   else                        /* now PC1 is change to low         */
      {
      CNT_L2++;
      }
   invert=invert^2;            /* to generate a high pulse         */
   }
if ((int_c&0x4)!=0)                    /* INT_CHAN_2 is active     */
   {
   if ((new_int_state&0x4)!=0)/* now PC2 is change to high         */
      {
      CNT_H3++;
      }
   else                        /* now PC2 is change to low         */
      {
      CNT_L3++;
      }
   invert=invert^4;            /* to generate a high pulse         */
   }
if ((int_c&0x8)!=0)                    /* INT_CHAN_3 is active     */
   {
   if ((new_int_state&0x8)!=0)/* now PC3 is change to high         */
      {
      CNT_H4++;
      }
   else                        /* now PC3 is change to low         */
      {
      CNT_L4++;
      }
   invert=invert^8;            /* to generate a high pulse         */
   }
now_int_state=new_int_state;
outportb(wBase+0x2a,invert);

if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```

WARRANTY/DISCLAIMER

OMEGA ENGINEERING, INC. warrants this unit to be free of defects in materials and workmanship for a period of **13 months** from date of purchase. OMEGA's WARRANTY adds an additional one (1) month grace period to the normal **one (1) year product warranty** to cover handling and shipping time. This ensures that OMEGA's customers receive maximum coverage on each product.

If the unit malfunctions, it must be returned to the factory for evaluation. OMEGA's Customer Service Department will issue an Authorized Return (AR) number immediately upon phone or written request. Upon examination by OMEGA, if the unit is found to be defective, it will be repaired or replaced at no charge. OMEGA's WARRANTY does not apply to defects resulting from any action of the purchaser, including but not limited to mishandling, improper interfacing, operation outside of design limits, improper repair, or unauthorized modification. This WARRANTY is VOID if the unit shows evidence of having been tampered with or shows evidence of having been damaged as a result of excessive corrosion; or current, heat, moisture or vibration; improper specification; misapplication; misuse or other operating conditions outside of OMEGA's control. Components which wear are not warranted, including but not limited to contact points, fuses, and triacs.

OMEGA is pleased to offer suggestions on the use of its various products. However, OMEGA neither assumes responsibility for any omissions or errors nor assumes liability for any damages that result from the use of its products in accordance with information provided by OMEGA, either verbal or written. OMEGA warrants only that the parts manufactured by it will be as specified and free of defects. OMEGA MAKES NO OTHER WARRANTIES OR REPRESENTATIONS OF ANY KIND WHATSOEVER, EXPRESS OR IMPLIED, EXCEPT THAT OF TITLE, AND ALL IMPLIED WARRANTIES INCLUDING ANY WARRANTY OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE HEREBY DISCLAIMED. LIMITATION OF LIABILITY: The remedies of purchaser set forth herein are exclusive, and the total liability of OMEGA with respect to this order, whether based on contract, warranty, negligence, indemnification, strict liability or otherwise, shall not exceed the purchase price of the component upon which liability is based. In no event shall OMEGA be liable for consequential, incidental or special damages.

CONDITIONS: Equipment sold by OMEGA is not intended to be used, nor shall it be used: (1) as a "Basic Component" under 10 CFR 21 (NRC), used in or with any nuclear installation or activity; or (2) in medical applications or used on humans. Should any Product(s) be used in or with any nuclear installation or activity, medical application, used on humans, or misused in any way, OMEGA assumes no responsibility as set forth in our basic WARRANTY/DISCLAIMER language, and, additionally, purchaser will indemnify OMEGA and hold OMEGA harmless from any liability or damage whatsoever arising out of the use of the Product(s) in such a manner.

RETURN REQUESTS/INQUIRIES

Direct all warranty and repair requests/inquiries to the OMEGA Customer Service Department. BEFORE RETURNING ANY PRODUCT(S) TO OMEGA, PURCHASER MUST OBTAIN AN AUTHORIZED RETURN (AR) NUMBER FROM OMEGA'S CUSTOMER SERVICE DEPARTMENT (IN ORDER TO AVOID PROCESSING DELAYS). The assigned AR number should then be marked on the outside of the return package and on any correspondence.

The purchaser is responsible for shipping charges, freight, insurance and proper packaging to prevent breakage in transit.

FOR **WARRANTY** RETURNS, please have the following information available BEFORE contacting OMEGA:

1. Purchase Order number under which the product was PURCHASED,
2. Model and serial number of the product under warranty, and
3. Repair instructions and/or specific problems relative to the product.

FOR **NON-WARRANTY** REPAIRS, consult OMEGA for current repair charges. Have the following information available BEFORE contacting OMEGA:

1. Purchase Order number to cover the COST of the repair,
2. Model and serial number of the product, and
3. Repair instructions and/or specific problems relative to the product.

OMEGA's policy is to make running changes, not model changes, whenever an improvement is possible. This affords our customers the latest in technology and engineering.

OMEGA is a registered trademark of OMEGA ENGINEERING, INC.

Where Do I Find Everything I Need for Process Measurement and Control? OMEGA...Of Course!

Shop online at www.omega.com

TEMPERATURE

- Thermocouple, RTD & Thermistor Probes, Connectors, Panels & Assemblies
- Wire: Thermocouple, RTD & Thermistor
- Calibrators & Ice Point References
- Recorders, Controllers & Process Monitors
- Infrared Pyrometers

PRESSURE, STRAIN AND FORCE

- Transducers & Strain Gages
- Load Cells & Pressure Gages
- Displacement Transducers
- Instrumentation & Accessories

FLOW/LEVEL

- Rotameters, Gas Mass Flowmeters & Flow Computers
- Air Velocity Indicators
- Turbine/Paddlewheel Systems
- Totalizers & Batch Controllers

pH/CONDUCTIVITY

- pH Electrodes, Testers & Accessories
- Benchtop/Laboratory Meters
- Controllers, Calibrators, Simulators & Pumps
- Industrial pH & Conductivity Equipment

DATA ACQUISITION

- Data Acquisition & Engineering Software
- Communications-Based Acquisition Systems
- Plug-in Cards for Apple, IBM & Compatibles
- Datalogging Systems
- Recorders, Printers & Plotters

HEATERS

- Heating Cable
- Cartridge & Strip Heaters
- Immersion & Band Heaters
- Flexible Heaters
- Laboratory Heaters

ENVIRONMENTAL MONITORING AND CONTROL

- Metering & Control Instrumentation
- Refractometers
- Pumps & Tubing
- Air, Soil & Water Monitors
- Industrial Water & Wastewater Treatment
- pH, Conductivity & Dissolved Oxygen Instruments

M4038/0104